

HTML

Structure, semantics, forms, accessibility and SEO basics

1

What is HTML?

Definition

HTML (HyperText Markup Language) describes the **structure and meaning** of a web page using elements (tags). The browser parses it into the **DOM tree**, CSS styles that tree and JavaScript makes it interactive.



How a page becomes pixels (the critical rendering path)

Minimal valid page

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <title>My page</title>
  </head>
  <body>
    <h1>Hello, world</h1>
    <p>A paragraph with a <a href="https://developer.mozilla.org">link</a>.
  </p>
  </body>
</html>
```



`<!DOCTYPE html>` switches the browser to **standards mode** (no quirks).

 `lang` helps screen readers and translation.

 The `viewport` meta tag makes the page responsive on phones.

Definition

Semantic elements describe **what content is**, not how it looks. They improve **accessibility**, **SEO** and readability.

Element	Use for
<code><header></code> / <code><footer></code>	Intro / closing content of a page or section
<code><nav></code>	Main navigation links
<code><main></code>	The unique main content (one per page)
<code><article></code>	Self-contained content: a post, a product card
<code><section></code>	A themed group of content with a heading
<code><aside></code>	Side content: related links, ads
<code><figure></code> + <code><figcaption></code>	Image/diagram with a caption
<code><button></code>	Actions (not a <code><div onclick></code>)

Common mistake

"Div soup": `<div class="button" onclick=...>` is not focusable and is not announced as a button. Use `<button>`; use `<a>` only for navigation to a URL.

Block vs inline, and common elements

Block	Inline
Starts on a new line, takes full width	Flows inside text, only as wide as content
<code>div</code> , <code>p</code> , <code>h1-h6</code> , <code>ul</code> , <code>section</code> , <code>form</code>	<code>span</code> , <code>a</code> , <code>strong</code> , <code>em</code> , <code>img</code> , <code>code</code> , <code>label</code>

- `<strong>` = important (bold), `<em>` = emphasis (italic) – semantic; `<b>` / `<i>` are only visual.
- `<img src alt>` – `alt` is required for accessibility; use `alt=""` for decorative images.
- Lists: `<ul>` unordered, `<ol>` ordered, `<li>` items, `<dl>` description list.
- Tables for tabular data only: `<table>` `<thead>` `<tbody>` `<tr>` `<th scope="col">` `<td>`.

Accessible form with built-in validation

```
<form id="signup" novalidate>
  <label for="email">Email</label>
  <input id="email" name="email" type="email" required autocomplete="email" />

  <label for="age">Age</label>
  <input id="age" name="age" type="number" min="18" max="99" />

  <label><input type="checkbox" name="terms" required /> I accept the
terms</label>
  <button type="submit">Sign up</button>
</form>
<p id="msg"></p>
<script>
  document.getElementById('signup').addEventListener('submit', (e) => {
    e.preventDefault()
    const form = e.target
    const msg = document.getElementById('msg')
    msg.textContent = form.checkValidity() ? 'Valid ✓' : 'Please fix the
highlighted fields'
    console.log(Object.fromEntries(new FormData(form)))
  })
</script>
```

- Every input needs a `<label>` (linked with `for` / `id` or by wrapping).
- Input types (`email`, `number`, `date`, `tel`, `url`) give free validation and the right mobile keyboard.
- Attributes: `required`, `min` / `max`, `minlength`, `pattern`, `autocomplete`.
- GET puts data in the URL (search, filters); POST sends it in the body (create, login).

Common mistake

Client-side validation is for UX only — always validate again on the server.

Accessibility (a11y)

- ✎ Use semantic elements first; **ARIA** only fills gaps ("no ARIA is better than bad ARIA").
- ✎ One `<h1>`, then headings in order (don't skip levels for styling).
- ✎ Everything clickable must work with the keyboard (Tab, Enter, Space) and show a visible focus ring.
- ✎ Text contrast at least 4.5:1; never use colour alone to carry meaning.
- ✎ `aria-label` names an icon-only button; `aria-live` announces dynamic updates.

Quick test: unplug the mouse and use the page with Tab only, then run Lighthouse / axe.

6

The <head>, SEO and loading scripts

- `<title>` and `<meta name="description">` show in search results.
- Open Graph tags (`og:title`, `og:image`) control link previews on social media.
- `<link rel="canonical">` avoids duplicate-content issues.
- Search engines understand semantic HTML and real links (`<a href>`).

Script tag	Downloads	Runs	Order kept?
<code><script></code>	blocks parsing	immediately	yes
<code><script defer></code>	in parallel	after HTML is parsed	yes
<code><script async></code>	in parallel	as soon as downloaded	no
<code><script type="module"></code>	in parallel	deferred by default	yes

Use `defer` for app scripts, `async` for independent ones (analytics). Add `loading="lazy"` on below-the-fold images.

Browser storage

	Capacity	Lifetime	Sent to server?
Cookie	~4 KB	Until expiry	Yes, on every request
localStorage	~5-10 MB	Until cleared	No
sessionStorage	~5 MB	Until the tab closes	No
IndexedDB	Large (quota)	Until cleared	No

Common mistake

Don't keep auth tokens in localStorage — any XSS can read them. Prefer `httpOnly`, `Secure`, `SameSite` cookies.

Revision

HTML = structure + meaning. Semantic tags, labelled forms, keyboard access, `defer` scripts, and the right storage for each job.

Q. What does `<!DOCTYPE html>` do?

A. Tells the browser to render in standards mode instead of quirks mode.

Q. Semantic vs non-semantic elements?

A. Semantic elements (header, nav, main, article, button) describe meaning; div and span carry none. Semantics help accessibility, SEO and maintenance.

Q. Block vs inline elements?

A. Block elements start on a new line and fill the width (div, p, h1). Inline elements flow in text and only take their content width (span, a, strong).

Q. `async` vs `defer` on script tags?

A. Both download in parallel. `defer` runs after parsing, in order. `async` runs as soon as it downloads, in any order.

Q. Why is the `alt` attribute important?

A. Screen readers read it, it shows if the image fails, and search engines use it. Decorative images get `alt=""`.

Q. `localStorage` vs `sessionStorage` vs cookies?

A. `localStorage` persists until cleared; `sessionStorage` lasts for the tab; cookies are small and are sent to the server on every request.

Q. How do you make a form accessible?

A. A label for every input, correct input types, clear error messages linked to fields, keyboard operable, and visible focus.

Q. What is the DOM?

A. The Document Object Model: the browser's tree of objects built from HTML that JavaScript can read and change.

Q. When should you use ARIA?

A. Only when native HTML can't express the role or state. Native elements first; ARIA fills gaps.

Q. What is the critical rendering path?

A. HTML → DOM, CSS → CSSOM, combined into the render tree, then layout and paint. Blocking CSS/JS delays the first paint.